

**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное
образовательное учреждение высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»**

Департамент цифровых, робототехнических систем и электроники

**«Работа со списками и кортежами в языке Python»
Отчет по лабораторной работе № 4
по дисциплине «Программирование на Python»
Вариант 5**

Выполнил студент группы ИВТ-б-о-24-1
Грабарь Артемий Павлович
«__» ноября 2025г.

Подпись студента _____
Работа защищена « » _____ 20__ г.
Проверил Воронкин Р.А. _____
(подпись)

Ставрополь 2025

Цель работы: приобретение навыков по работе со списками и кортежами при написании программ с помощью языка программирования Python версии 3.x.

Ссылка на GitHub: https://github.com/Arhi258/Laba_4.git

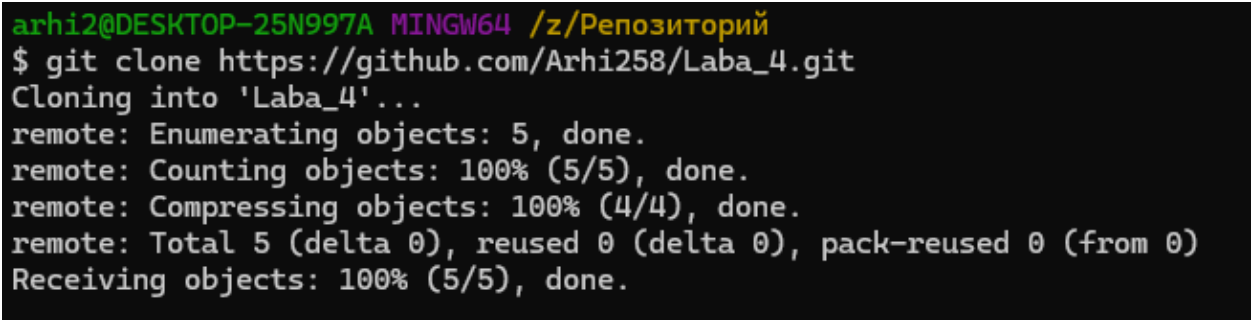
Задание:

1. Создать общедоступный репозиторий на GitHub, котором будет использована лицензия MIT и язык программирования Python.
2. Выполнить клонирование данного репозитория.
3. Дополнить файл .gitignore необходимыми правилами для работы с IDE PyCharm.
4. Создать проект PyCharm в папке репозитория.
5. Проработать примеры лабораторной работы. Создать для каждого примера отдельный модуль языка Python. Зафиксировать изменения в репозитории.
6. Привести в отчете скриншоты результатов выполнения каждой из программ примеров при различных исходных данных вводимых с клавиатуры.
7. Привести в отчете скриншоты работы программ решения индивидуальных заданий. Задание 1: Ввести список A из 10 элементов, найти сумму элементов, больших 3 и меньших 8 и вывести ее на экран. Задание 2: В списке, состоящем из вещественных элементов, вычислить: максимальный элемент списка; сумму элементов списка, расположенных до последнего положительного элемента. Сжать список, удалив из него все элементы, модуль которых находится в интервале $[a, b]$. Освободившиеся в конце списка элементы заполнить нулями. Задание 3: Если в кортеже есть хотя бы одна пара одинаковых соседних элементов, то напечатать все элементы, следующие за элементами первой из таких пар.
8. Зафиксировать сделанные изменения в репозитории.
9. Добавить отчет по лабораторной работе в формате PDF в папку doc репозитория. Зафиксировать изменения.

10. Отправить сделанные изменения на сервер GitHub.

Ход работы:

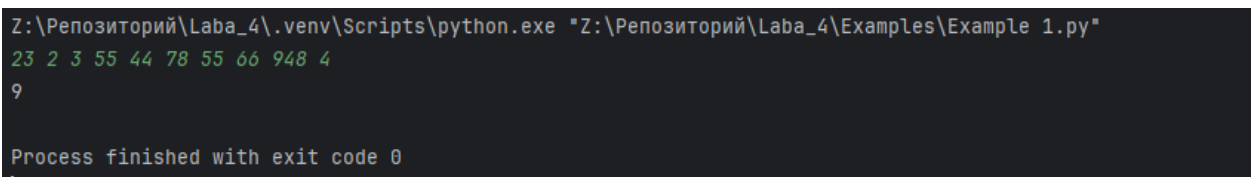
1. Был создан общедоступный репозиторий на GitHub, в котором использована лицензия MIT и язык программирования Python.
2. Было выполнено клонирование репозитория (рис 1).



```
arhi2@DESKTOP-25N997A MINGW64 /z/Репозиторий
$ git clone https://github.com/Arhi258/Laba_4.git
Cloning into 'Laba_4'...
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 5 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
Receiving objects: 100% (5/5), done.
```

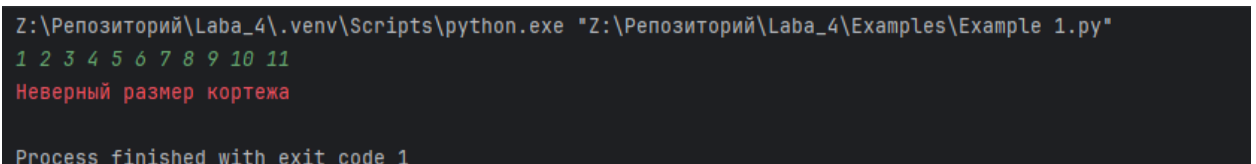
Рисунок 1. Клонирование репозитория

3. Файл .gitignore был дополнен необходимыми правилами для работы с IDE PyCharm.
4. В папке репозитория был создан проект PyCharm.
5. Были проработаны примеры, приведенные в лабораторной работе. Для каждого примера был создан отдельный модуль Python. Изменения были зафиксированы в репозитории.
6. Каждый из примеров был проверен на работоспособность при различных исходных данных вводимых с клавиатуры (рис 2, 3, 4, 5). Полный исходный код программы приведен в Приложении 1, 2.



```
Z:\Репозиторий\Laba_4\.venv\Scripts\python.exe "Z:\Репозиторий\Laba_4\Examples\Example 1.py"
23 2 3 55 44 78 55 66 948 4
9
Process finished with exit code 0
```

Рисунок 2. Работа программы с кортежем при верном размере



```
Z:\Репозиторий\Laba_4\.venv\Scripts\python.exe "Z:\Репозиторий\Laba_4\Examples\Example 1.py"
1 2 3 4 5 6 7 8 9 10 11
Неверный размер кортежа
Process finished with exit code 1
```

Рисунок 3. Работа программы с кортежем при неверном размере

```
Z:\Репозиторий\Laba_4\.venv\Scripts\python.exe "Z:\Репозиторий\Laba_4\Examples\Example 1_2.py"
1 2 3 4 5 6 7 8 9 10
10
Process finished with exit code 0
```

Рисунок 4. Работа программы со списком при верном размере

```
Z:\Репозиторий\Laba_4\.venv\Scripts\python.exe "Z:\Репозиторий\Laba_4\Examples\Example 1_2.py"
1 1 1 1
Неверный размер списка
Process finished with exit code 1
```

Рисунок 5. Работа программы со списком при неверном размере

7. Были выполнены индивидуальные задания, скриншоты работы программ прилагаются (рис 6, 7, 8). Полный исходный код программы приведен в Приложении 3, 4, 5 соответственно.

```
Z:\Репозиторий\Laba_4\.venv\Scripts\python.exe "Z:\Репозиторий\Laba_4\Tasks\Task 1.py"
Введите 10 элементов списка: 1 2 3 4 5 6 7 8 9 10
Сумма элементов больших 3 и меньших 8 равна: 22
Process finished with exit code 0
```

Рисунок 6. Результат выполнения программы для задания 1

```
Введите элементы списка: 1 2 3 0.7 -14 25 -6 -0.78 0.3
Введите значения для сжатия списка:
Левая граница списка: 2
Правая граница списка: 12
Максимальный элемент списка до сжатия: 25.0
Сумма элементов списка до сжатия, расположенных до последнего положительного элемента: 11.22
Сжатый список: [1.0, 0.7, -14.0, 25.0, -0.78, 0.3, 0.0, 0.0, 0.0]
```

Рисунок 7. Результат выполнения программы для задания 2

```
Z:\Репозиторий\Laba_4\.venv\Scripts\python.exe "Z:\Репозиторий\Laba_4\Tasks\Task 3.py"
Введите кортеж: 1 2 33 4 5 5 6 6 6 7 7 7 8 99 99
Элементы после первой пары одинаковых чисел:(6, 6, 6, 7, 7, 7, 8, 99, 99)
```

Рисунок 8. Результат выполнения программы для задания 3

8. Все изменения были зафиксированы в репозитории (рис 9).

```
arhi2@DESKTOP-25N997A MINGW64 /z/Репозиторий/Laba_4 (main)
$ git add .

arhi2@DESKTOP-25N997A MINGW64 /z/Репозиторий/Laba_4 (main)
$ git commit -m "Выполнены индивидуальные задания"
[main 93d693e] Выполнены индивидуальные задания
3 files changed, 86 insertions(+)
create mode 100644 Tasks/Task 1.py
create mode 100644 Tasks/Task 2.py
create mode 100644 Tasks/Task 3.py
```

Рисунок 9. Зафиксированные изменения

9. В папку doc был добавлен отчет по лабораторной работе в формате PDF. Изменения были зафиксированы.

10. Сделанные изменения были отправлены на сервер GitHub.

Контрольные вопросы:

1. Списки в языке Python – изменяемые последовательности элементов, которые можно использовать для хранения различных данных.
2. Создание списка осуществляется с помощью [] или функции list().
3. Списки в оперативной памяти хранятся, используя структуры данных, которые организуют элементы в виде связанных объектов или непрерывных блоков памяти.
4. Перебор элементов списка выполняется с помощью цикла.
5. Арифметические операции со списками: + и *.
6. Чтобы проверить, есть ли заданный элемент в списке необходимо использовать оператор in (пример: if a in lst). Если нужно, чтобы элемент отсутствовал в списке, используется оператор not in.
7. Для определения числа вхождений используется функция count().
8. Для вставки элемента в список используется метод insert, для добавления элемента в список используется метод append.
9. Для сортировки списка используется метод sort.
10. Удалить элемент списка можно, написав его индекс в методе pop, если не указывать индекс, то удалится последний элемент. Элемент по значению можно удалить с помощью метода remove. Оператор del можно

использовать для удаления элемента из списка или для среза. Чтобы удалить все элементы можно использовать `clear`.

11. Списковое включение является частью синтаксиса языка, которая представляет простой способ построения списков (Пример: `a = [i for i in range(int(input()))]`). Списковое включение позволяет обработать список без использования таких функций как `map` и `filter`.

12. Срезы позволяют быстро и лаконично решить задачи выборки элементов, он задается тройкой чисел `start:stop:step`.

13. Функции агрегации: `len(l)` – получить число элементов в списке `l`. `min(l)` – получить минимальный элемент списка `l`. `max(l)` – получить максимальный элемент списка `l`. `sum(l)` – получить сумму элементов списка `l`, если список `l` содержит только числовые значения.

14. Для создания копии списка необходимо использовать либо метод `copy`, либо оператор среза.

15. Функция `sorted()` создает новый отсортированный список из элементов любого итерируемого объекта, не изменяя исходный объект.

16. Кортеж (`tuple`) – неизменяемая структура данных, очень похожая на список.

17. Кортежи используются для того, чтобы обезопасить данные от случайного изменения. Так же кортежи занимают меньше места и работают быстрее чем списки.

18. Для создания кортежей можно воспользоваться следующими способами: `a = ()`, `a = tuple([1,2,3,4])`.

19. Доступ к элементам кортежа осуществляется через индекс.

20. Деструктуризация кортежа нужна для упрощения работы с кортежами.

21. Благодаря тому, что кортежи удобно разбирать и собирать, удобно делать множественное присваивание `((a, b) = (b, a))` – были присвоены в `a` и `b` значения из кортежа, состоящего из значений переменных `b` и `a`).

22. Для того, чтобы выбрать элементы кортежа с помощью среза строится конструкция: $T2 = T1[i:j]$ где, $T2$ – новый кортеж; $T1$ – исходный кортеж; i, j – нижняя и верхняя граница среза, значение j определяет позицию за последним элементом среза.

23. Операция конкатенации для кортежей выполняется следующим образом: $T3 = T2 + T1$ где, $T3$ – кортеж, являющийся результатом; $T1, T2$ – кортежи, для которых выполняется конкатенация. Повторение кортежей выполняется следующим образом: $T2 = T1 * n$ где $T2$ – результирующий кортеж; $T1$ – исходный кортеж, который нужно повторить n раз; n – количество повторений кортежа $T1$.

24. Обход элементов кортежа можно выполнить с помощью цикла `while` или `for`.

25. Операция `in` позволяет проверить принадлежность элемента кортежу.

26. Методы работы с кортежами: метод `index()` – поиск позиции элемента в кортеже; метод `count()` – количество вхождений элемента в кортеж; метод.

27. Использование функций агрегаций (таких как `len()`, `sum()` и т.д.) допустимо при работе с кортежами.

28. Напрямую создать кортеж с помощью спискового включения нельзя, т.к. синтаксис всегда создает список, а не кортеж. Чтобы получить кортеж из спискового включения, нужно явно обернуть списковое включение `tuple()`.

Вывод: в результате работы были получены навыки по работе со списками и кортежами при написании программ с помощью языка программирования Python версии 3.x.

Приложения.

Приложение 1. Исходный код для примера 1:

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

import sys
```

```

if __name__ == '__main__':
    # Ввести кортеж одной строкой.
    A = tuple(map(int, input().split()))
    # Проверить количество элементов кортежа.
    if len(A) != 10:
        print("Неверный размер кортежа", file=sys.stderr)
        exit(1)

    # Найти искомую сумму.
    s = 0
    for item in A:
        if abs(item) < 5:
            s += item

    print(s)

```

Приложение 2. Исходный код для примера 2:

```

#!/usr/bin/env python3
# -*- coding: utf-8 -*-

import sys

if __name__ == '__main__':
    # Ввести список одной строкой.
    A = list(map(int, input().split()))
    # Проверить количество элементов списка.
    if len(A) != 10:
        print("Неверный размер списка", file=sys.stderr)
        exit(1)

    # Найти искомую сумму.
    s = sum(a for a in A if abs(a) < 5)
    print(s)

```

Приложение 3. Исходный код для задания 1:

```

#!/usr/bin/env python3
# -*- coding: utf-8 -*-

import sys

if __name__ == '__main__':
    # Ввод списка
    a_list = list(map(int, input("Введите 10 элементов списка: ").split()))

```

```

# Проверка размера списка
if len(a_list) != 10:
    print("Неверный размер списка", file=sys.stderr)
    exit(1)

s = 0
for a in a_list:
    if 3 < a < 8:
        s += a

print(f"Сумма элементов больших 3 и меньших 8 равна: {s}")

```

Приложение 4. Исходный код для задания 2:

```

#!/usr/bin/env python3
# -*- coding: utf-8 -*-

import sys

if __name__ == '__main__':
    # Ввод элементов списка
    a_list = list(map(float, input("Введите элементы списка: ").split()))

    # Границы сжатия списка
    print("Введите значения для сжатия списка: ")
    left_border = float(input("Левая граница списка: "))
    right_border = float(input("Правая граница списка: "))

    # Проверка корректности границ
    if left_border >= right_border or left_border < 0 or right_border < 0:
        print("Неверно введенные границы", file=sys.stderr)
        exit(1)

```

```

print(f"Максимальный элемент списка до сжатия: {max(a_list)}")

# Вычисление индекса последнего положительного элемента
stop_index = len(a_list) - 1
for item in reversed(a_list):
    if item > 0:
        break
    stop_index -= 1

# Вычисление суммы списка до последнего положительного элемента
total_sum = sum(a_list[:stop_index + 1])
print(
    f"Сумма элементов списка до сжатия, расположенных до "
    f"последнего положительного элемента: {total_sum}"
)

# Цикл сжатия списка
n = len(a_list)
processed = 0
i = 0
while processed < n:
    if left_border <= abs(a_list[i]) <= right_border:
        a_list.pop(i)
        a_list.append(0.0)
    else:
        i += 1
    processed += 1

print(f"Сжатый список: {a_list}")

```

Приложение 5. Исходный код для задания 3:

```
#!/usr/bin/env python3
```

```
# -*- coding: utf-8 -*-

import sys

if __name__ == '__main__':
    # Ввод кортежа
    a_tuple = tuple(map(int, input("Введите кортеж: ").split()))

    # Вычисление индекса начала вывода
    start_idx = 0
    for i in range(len(a_tuple) - 1):
        if a_tuple[i] == a_tuple[i + 1]:
            start_idx = i + 2
            break

    print(
        f"Элементы после первой пары одинаковых чисел:"
        f"{a_tuple[start_idx:]}"
    )
```